

De IntRoLab.

Français ▼

Sommaire

- 1 Introduction
 - 1.1 Why do we need new motor controllers
- 2 Schedule
- 3 Team
- 4 Documents
- 5 Arm Improvements
 - 5.1 Motor PCB
 - 5.2 Motor Controllers
 - 5.3 Cabling Improvements
 - 5.4 Pitch motor coupling updated
- 6 Head Improvements
 - 6.1 Motor controllers for head movement (Yaw + Pitch)
- 7 Tuning the motor controller
- 8 RS-232 protocol enhancements
 - 8.1 CAN command format for RS-232 communication
 - 8.2 Motor Controller Variables
- 9 MRA, MLA, MRN Modified Serial Messages
 - 9.1 MRA - Move Right Arm (New version)
 - 9.2 MLA - Move Left Arm (New version)
 - 9.3 MRN - Move Robot Neck (New version)
- 10 New Joints Configuration
 - 10.1 Useful Examples Using Direct CAN Commands
 - 10.1.1 Read all positions from the motor controller
 - 10.1.2 Read all speeds from the motor controller
 - 10.1.3 Read all SetPoints from the motor controller
 - 10.1.4 Write the SetPoint of motor [0-3]
- 11 Additional Notes & Comments

Introduction

Some problems have been experienced with the initial Reddy configuration :

1. Motors can overheat and "self-destruct"
2. Motors are noisy because of the switching frequency of the original servo controllers
3. No position feedback on servo motors is available
4. No current control on servo motors is available
5. No Speed profile on servo motors is available
6. The head will fall when no power is applied (the mechanism may break over time)
7. Maintaining the head position requires more power than necessary because of the initial mechanism.

This wiki page describes what have been done to minimize those problems. **Most of the changes will be invisible to the end user.** Changes in the initial serial protocol are adding functionality and can be used if the user requires more control of the motor controllers. Expert users will have the possibility to change the motor controller behavior if required. **The only change that will be required in the user code is the range of the variables to control the joints.** This is the result of changing the hardware for position acquisition that is now done using 12bits ADC converters. The MLA, MRA and MRN are then modified to fit the new motor controllers and now uses 12 bits position comands instead of a position and the speed. Please see this section for more information on that matter.

Why do we need new motor controllers

It was a mistake to use standard servo motors for Reddy because :

- The datasheet of the motor is not very clear about the continuous torque they can provide
 - This caused us a lot of problems, because **we are using the servo motors with torque they cannot sustain for a long period of time.** This causes overheat in the motor winding and can destroy the servo motor.
- Noise caused by the original switching frequency was neglected. The switching frequency is about 300Hz and is very audible.
- The real positions of the joints are not available.
 - They can be approximated with the last setpoint, but it is not safe to assume that.

The best solution to fix those problems is to change all the problematic motors (8 motors in fact) with motors (+ gearboxes) with additional torque and adequate motor controllers. This would require a complete change in the mechanical design of the robot. This solution is not a "low cost" option and was not considered.

The best solution we have to compensate for the motor problems is to create a new motor controller that have the following characteristics :

- Cost < 1000\$ for all motors.
- Can monitor the motor temperature and shutdown if overheating
- Can provide position feedback
- Can monitor the motor current and limit the current to avoid breaking the hardware
- Can provide position control with speed profile to avoid breaking the hardware
 - The result is that the motor position control will be slower, but safer
- Can be eventually used if motors are replaced with motors with greater torque

We would have liked to place the new motor controllers directly in the servo motor housing, but it was not possible because of space constraints. Also, making the motor controller only work for servo motors would limit the reusability of the controller. We have finally come to the following solution :

- Place a small PCB into the motor housing that will be used for thermal monitoring and connecting the motor power and potentiometer.
- Create a master controller installed in Reddy's torso that is easily accessible and programmable.

Schedule

| | |
|---------------------------------------|---|
| Week 31, July 27 - August 1 | <ul style="list-style-type: none"> ▪ Ordered new coupling for shoulder ▪ Fab. + installation of head mechanism for gravity assist (previous one had interference problems with other head parts) ▪ Assembly and installation of 1 motor PCB for neck ▪ Assembly and installation of the speaker PCB + support ▪ Final tuning of motor controllers ▪ Documentation (protocol) |
| Week 32, August 3 - August 9 | <ul style="list-style-type: none"> ▪ Final tests & tuning <ul style="list-style-type: none"> ▪ <i>We are still waiting for motor couplings and speaker PCB. This will delay the tests until we receive them.</i> ▪ Update1 : We did receive the speaker PCB. Still waiting for the couplings. ▪ Update2 : (08/07/2009) Just received the motor couplings. ▪ Update3 : (08/14/2009) Couplings are installed |
| Week 33, August 10 - August 15 | <ul style="list-style-type: none"> ▪ Install motor couplings ▪ Schedule a meeting for robot exchange <ul style="list-style-type: none"> ▪ <i>If speaker PCB & coupling is received.</i> |

Team

- Marc-Antoine Legault
- Serge Caron
- Jean-François Duval
- Alexis Demers
- Dominic Létourneau (mailto:Dominic.Letourneau@USherbrooke.ca)
- François Michaud

Documents

- File:MotorControllerTuningWin32.rar
- Source code will be available soon
- File:ReddyMotorDriveSchematic.pdf
- File:ReddyMotorInternalPCB.pdf

Arm Improvements

To have better control of the arm (and neck) motors, we did the following :

- Change the Motor PCB to support
 - Temperature detector. A signal will be sent to the motor controller if the motor temperature is higher than 60 degrees Celsius. Analog temperature value is also available.
 - Potentiometer feedback.
 - Motor power connection.
- Design a new PCB for motor control with support for 4 motors
- Communicate with the CAN bus already available on Reddy
- Allow position feedback from the RS-232 interface
- Create a tuning GUI for easier setup of the robot controllers

Motor PCB

Modified Servo Motors



Motor Controllers



- A new motor drive that can control up to 4 motors have been designed.
- New dsPIC30F code for position control with current limit, trapezoidal speed, temperature limit
- A new adjustable bracket for fixing the drive to the side of the robot have been designed.
- We installed two motor controllers (one on each side) to allow the modification of 8 motors. Motors modified are :
 - Arm-Elbow (x2)
 - Arm-Shoulder-Yaw (x2)
 - Arm-Shoulder-Pitch (x2)
 - Neck-Yaw (x1)
 - Neck-Pitch (x1)

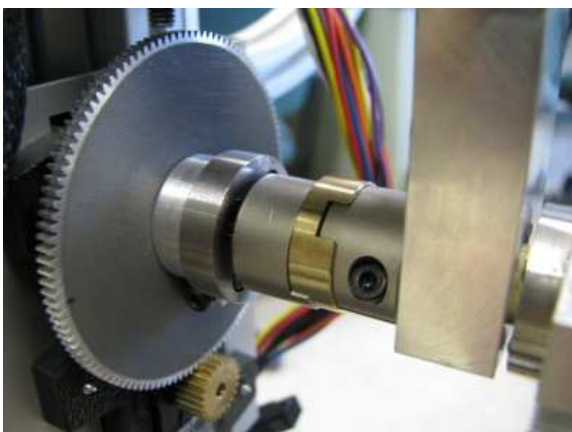
Cabling Improvements



- Cables are now better organized
- Added connectors inside the arm to disconnect motors easily for repairs
- Signals are color coded as follow (Number in brackets is the pin number of the motor side connector)
 - Red = Power 5V [5]
 - Black = Power GND [6]
 - Yellow = Motor A [10]
 - Brown = Motor B [9]
 - Blue = Analog temperature readout (refer to LM20 datasheet) [7]
 - Orange = Over temperature flag (5V if more than 60 degrees) [8]
 - Gray = Potentiometer GND (tied with Power GND) [4]
 - White = Potentiometer 5V (tied with Power 5V) [3]
 - Purple = Potentiometer signal (0-5V analog) [2]

Pitch motor coupling updated

We replaced the motor coupling on both arms with a bigger one that can sustain more torque on the motor shaft.



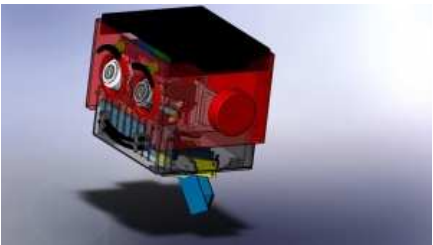
Head Improvements

We added gravity compensation for the head yaw to minimize power required for motors.

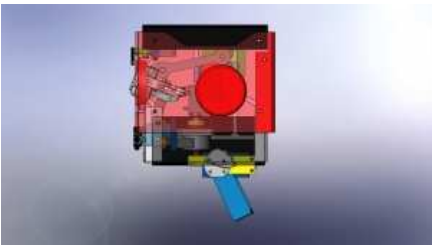
Head modifications



The mechanical part that was modified



View of the head with new part added (1)



View of the head with new part added (2)

This will also prevent the head from falling when the robot is turned off.

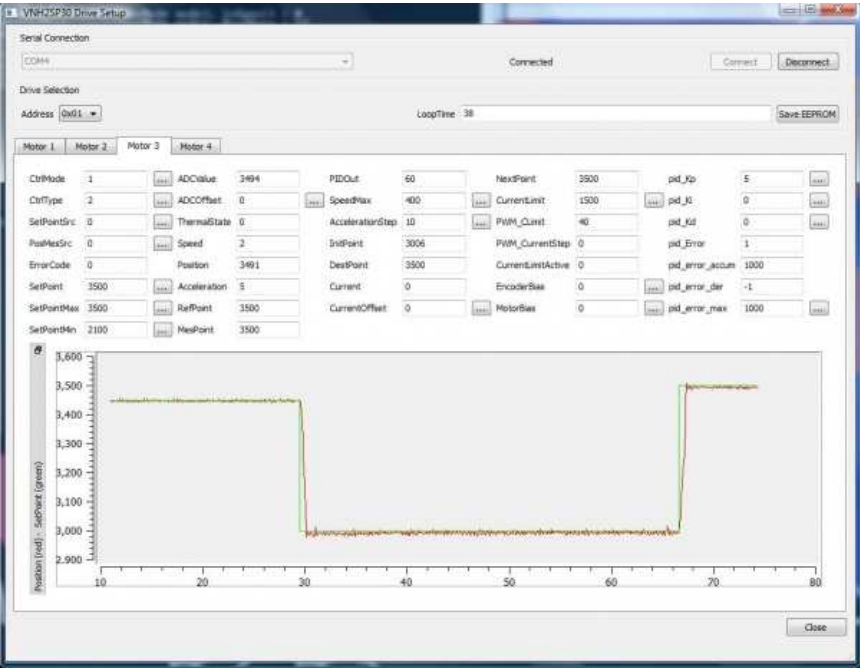
Motor controllers for head movement (Yaw + Pitch)

- Same motor controllers were used for the neck to add overheating protection and better feedback / control
- Internal servo potentiometers were used for position control.

Tuning the motor controller

A GUI was created to facilitate motor tuning.

Warning! This GUI is intended for Engineers at IntRoLab and changing control values can lead to undesirable effects. Assistance will be required before changing controller values.



RS-232 protocol enhancements

- We added a CAN command to the RS-232 robot interface to allow direct communication with the motor controllers. The RS-232 robot interface now acts as a RS-232 to CAN bridge and bidirectional communication is now possible. This will allow feedback from the motor controllers (current, position, speed, etc.)

```
(RS-232 CAN COMMAND) <-----> (MOTOR DRIVE CAN INTERFACE)

* Serial Communication is at 57600 baud, 8 bits, no parity
```

CAN command format for RS-232 communication

Serial messages containing CAN commands have the following stucture :

| RS-232 Data Format (16 bytes total) (part 1) | | | |
|--|---------------------|--------------|-------------------------|
| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |

| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
|---|--|--|--|
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> We had to change the standard protocol here to address more than 256 bytes of internal memory of the motor controllers. MSB of the MSG_CMD will be stored here (3 bits max). | <ul style="list-style-type: none"> CAN_TYPE_EMERGENCY=0x01 CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 CAN_TYPE_SENSOR_HIGH_PRIORITY=0x04 CAN_TYPE_ACTUATOR_LOW_PRIORITY=0x08 CAN_TYPE_SENSOR_LOW_PRIORITY=0x10 CAN_TYPE_REQUEST_DATA=0x20 CAN_TYPE_USER2=0x40 CAN_TYPE_EVENTS=0x80 | <ul style="list-style-type: none"> bit 0-3 = DATA_LENGTH (0-8) bit 4 = RTR bit 5 = Boot (READ=1/WRITE=0) bit 6 = Boot (EEPROM=1/RAM=0) |

RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|--|---|---|
| (1 byte) | (1 byte) | (8 bytes) |
| <pre>if(MSG_TYPE == REQUEST) ■ CMD = RAM OFFSET FOR READ/WRITE endif if(MSG_TYPE == ACTUATOR_HIGH_PRIORITY) ■ CAN_ACTUATOR_CMD_POSITIONS=0x90 ■ CAN_ACTUATOR_CMD_SPEEDS=0x91 ■ CAN_ACTUATOR_CMD_SETPOINTS=0x92 ■ CAN_ACTUATOR_CMD_WRITE_SETPOINT=0x93 endif</pre> | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> Will depend on commands |

To keep things simple, we are using fixed length messages with 16 bytes. In some cases, the bytes containing the CAN frame data are transmitted but are ignored by the RS-232 Server.

Note: It is not essential to understand the structure of the CAN messages. Source code will be provided.

Motor Controller Variables

Motor controller internal variables can be represented like this :

```
typedef struct _SharedVariables
{
    union {
        struct {
            unsigned char CtrlMode;
            unsigned char CtrlType;
            unsigned char SetPointSource;
            unsigned char PosMesSource;
            unsigned short ErrorCode;
            short SetPoint;
            short SetPointMax; //Maximum SetPoint
            short SetPointMin; //Minimum SetPoint
            short Current;
            short CurrentOffset;
            short ADCValue;
            short ADCOffset;
            short ICValue;
            short ICOffset;
            short ThermalState;
            //CONTROL VARIABLES
            short Speed; //16 bit speed
            short Position; //16 bits position
            short Acceleration;
            //PID REF AND MES POINT
            short RefPoint;
            short MesPoint;
            float pid_kp;
            float pid_ki;
            float pid_kd;
            float pid_error;
            float pid_error_accum;
            float pid_error_derivative;
            float pid_error_accum_max; //PID I value max limit (abs)
            short PIDOut;
            short SpeedMax; //for Trapz
            short AccelerationStep; //for Trapz, Speed
            short InitPoint; //for Trapz
            short DestPoint; //for Trapz
            short NextPoint; //for Trapz
            short CurrentLimit; //unit = mA
            short PWM_CurrentLimit;
            short PWM_CurrentStep; //PWM increment
            unsigned char CurrentLimitActive;
        };
    };
};
```

```

        unsigned char EncoderBias; //Setting this to 1 will invert the encoder
        unsigned char MotorBias; //Setting this to 1 will invert motor direction
        unsigned char padding[3];
    } __attribute__((packed));

    unsigned char m_data[];
};
} SharedVariables;

typedef struct _GlobalVariables
{
    union {
        struct {

            //All control variables
            SharedVariables m_variables[4];

            //additional (global) variables
            unsigned char m_ESTOPEnabled;
            unsigned char m_WriteEEPROM;
            short m_loopTime;
            unsigned char m_padding[4];

        } __attribute__((packed));

        unsigned char m_data[];
    }; //union
} GlobalVariables;

```

Notes :

- Understanding the structure of the controller is not required for normal operation.
- The structure GlobalVariables represents the memory organization of the motor controller. You can see that it contains control variables (SharedVariables structure) for 4 motor axis.
- Using the CAN protocol, we can access each variable and read/write its value.
- The GUI was designed for easy tuning of those variables by our engineers.

MRA, MLA, MRN Modified Serial Messages

Those commands are still available, but have been modified to fit the new motor controllers :

MRA - Move Right Arm (New version)

Previous serial message format was :

- Header[0] = 'm'
- Header[1] = 'r'
- Header[2] = 'a'
- Param[0] = Desired right elbow pitch position
- Param[1] = Desired right elbow pitch speed
- Param[2] = Desired right shoulder yaw position
- Param[3] = Desired right shoulder yaw speed
- Param[4] = Desired right shoulder pitch position
- Param[5] = Desired right shoulder pitch speed

The new MRA serial message format is :

- Header[0] = 'm'
- Header[1] = 'r'
- Header[2] = 'a'
- Param[0] = Desired right elbow pitch position LSB
- Param[1] = Desired right elbow pitch position MSB
- Param[2] = Desired right shoulder yaw position LSB
- Param[3] = Desired right shoulder yaw position MSB
- Param[4] = Desired right shoulder pitch position LSB
- Param[5] = Desired right shoulder pitch position MSB

Note : Speed was optimized to for fastest (safe) movement possible with a trapezoidal speed

MLA - Move Left Arm (New version)

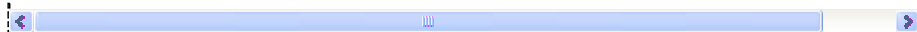
Previous serial message format was :

- Header[0] = 'm'
- Header[1] = 'l'
- Header[2] = 'a'
- Param[0] = Desired left elbow pitch position
- Param[1] = Desired left elbow pitch speed
- Param[2] = Desired left shoulder yaw position
- Param[3] = Desired left shoulder yaw speed
- Param[4] = Desired left shoulder pitch position
- Param[5] = Desired left shoulder pitch speed

The new MLA serial message format is :

- Header[0] = 'm'
- Header[1] = 'l'
- Header[2] = 'a'
- Param[0] = Desired left elbow pitch position LSB
- Param[1] = Desired left elbow pitch position MSB
- Param[2] = Desired left shoulder yaw position LSB
- Param[3] = Desired left shoulder yaw position MSB
- Param[4] = Desired left shoulder pitch position LSB
- Param[5] = Desired left shoulder pitch position MSB

Note : Speed was optimized to for fastest (safe) movement possible with a trapezoidal speed



MRN - Move Robot Neck (New version)

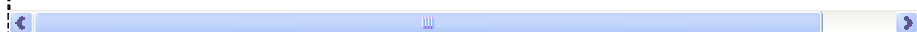
Previous serial message format was :

- Header[0] = 'm'
- Header[1] = 'r'
- Header[2] = 'n'
- Param[0] = Desired neck pitch position
- Param[1] = Desired neck pitch speed
- Param[2] = Desired neck yaw position
- Param[3] = Desired neck yaw speed

The new MRN serial message format is :

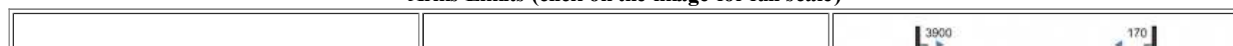
- Header[0] = 'm'
- Header[1] = 'r'
- Header[2] = 'n'
- Param[0] = Desired neck pitch position LSB
- Param[1] = Desired neck pitch position MSB
- Param[2] = Desired neck yaw position LSB
- Param[3] = Desired neck yaw position MSB

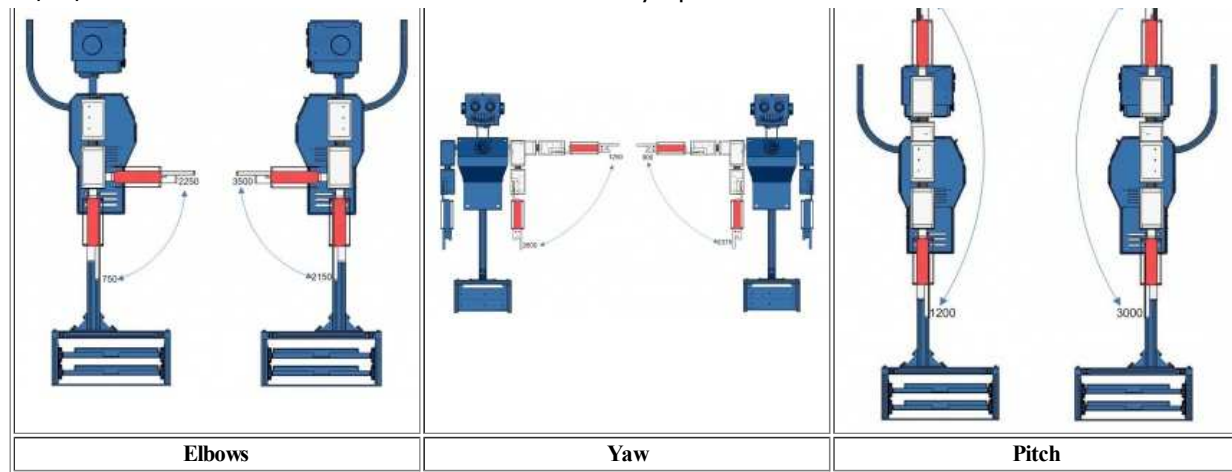
Note : Speed was optimized to for fastest (safe) movement possible with a trapezoidal speed



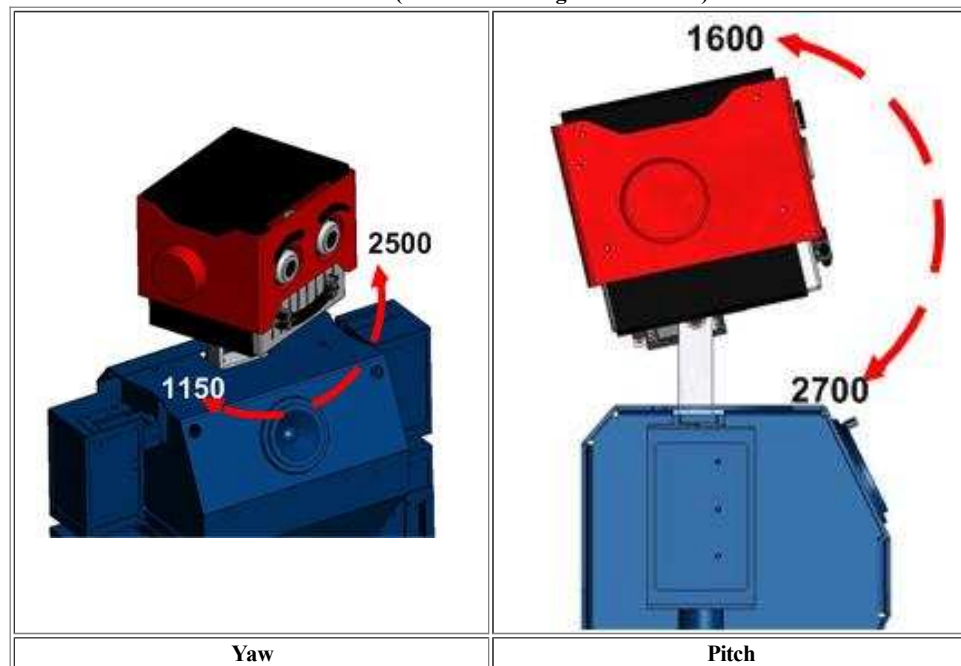
New Joints Configuration

Arms Limits (click on the image for full scale)





Neck Limits (click for the image for full scale)



Updated Reddy Joints Configuration

| Motor controller id=0x01 | Motor controller id=0x02 |
|---|--|
| <ul style="list-style-type: none"> Motor 0 = Left Arm Shoulder Pitch <ul style="list-style-type: none"> Range: Motor 1 = Left Arm Shoulder Yaw <ul style="list-style-type: none"> Range: Motor 2 = Left Arm Elbow <ul style="list-style-type: none"> Range: Motor 3 = Head Yaw ("NO") <ul style="list-style-type: none"> Range: | <ul style="list-style-type: none"> Motor 0 = Right Arm Shoulder Pitch <ul style="list-style-type: none"> Range: Motor 1 = Right Arm Should Yaw <ul style="list-style-type: none"> Range: Motor 2 = Right Arm Elbow <ul style="list-style-type: none"> Range: Motor 3 = Yeard Pitch ("YES") <ul style="list-style-type: none"> Range: |

Useful Examples Using Direct CAN Commands

Those examples will probably contain all you will need to do to operate the robot.

Read all positions from the motor controller

The computer sends this to the RS-232 server (Remote request) :

Read all positions from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|-----------|---------------------|--------------|-------------------------|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |

| | | | |
|---|---|--|---|
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> bit 0-3 = 8 bit 4 = RTR = 1 bit 5 =1 (not used) bit 6 =1 (not used) |
|---|---|--|---|

Read all positions from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|---|---|---|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> CAN_ACTUATOR_CMD_POSITIONS=0x90 | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> CAN_MSG_DATA[0-7] (ignored) |

Serial Message Sent:

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x78
- serial_message[6] = 0x90
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = (ignored)
- serial_message[9] = (ignored)
- serial_message[10] = (ignored)
- serial_message[11] = (ignored)
- serial_message[12] = (ignored)
- serial_message[13] = (ignored)
- serial_message[14] = (ignored)
- serial_message[15] = (ignored)

The RS-232 server responds (response to remote request):

Read all positions from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|---|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> bit 0-3 = 8 bit 4 = RTR = 0 bit 5 =1 (not used) bit 6 =1 (not used) |

Read all positions from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|---|---|---|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> CAN_ACTUATOR_CMD_POSITIONS=0x90 | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> CAN_MSG_DATA[0-1] <ul style="list-style-type: none"> LSB, MSB Position motor 0 CAN_MSG_DATA[2-3] <ul style="list-style-type: none"> LSB, MSB Position motor 1 CAN_MSG_DATA[4-5] <ul style="list-style-type: none"> LSB, MSB Position motor 2 |

| | | |
|--|--|--|
| | | <ul style="list-style-type: none"> ■ CAN_MSG_DATA[6-7] ■ LSB, MSB Position motor 3 |
|--|--|--|

Serial Message Received :

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x68
- serial_message[6] = 0x90
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = Motor 0 Position LSB
- serial_message[9] = Motor 0 Position MSB
- serial_message[10] = Motor 1 Position LSB
- serial_message[11] = Motor 1 Position MSB
- serial_message[12] = Motor 2 Position LSB
- serial_message[13] = Motor 2 Position MSB
- serial_message[14] = Motor 3 Position LSB
- serial_message[15] = Motor 3 Position MSB

Read all speeds from the motor controller

The computer sends this to the RS-232 server (Remote request) :

Read all currents from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|---|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> ■ header[0] = 'c' ■ header[1] = 'a' ■ header[2] = 'n' | <ul style="list-style-type: none"> ■ CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> ■ CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> ■ bit 0-3 = 8 ■ bit 4 = RTR = 1 ■ bit 5 =1 (not used) ■ bit 6 =1 (not used) |

Read all currents from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|--|---|--|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> ■ CAN_ACTUATOR_CMD_SPEEDS=0x91 | <ul style="list-style-type: none"> ■ 0x01 = First controller ■ 0x02 = Second controller | <ul style="list-style-type: none"> ■ CAN_MSG_DATA[0-7] (ignored) |

Serial Message Sent:

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x78
- serial_message[6] = 0x91
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = (ignored)
- serial_message[9] = (ignored)
- serial_message[10] = (ignored)
- serial_message[11] = (ignored)
- serial_message[12] = (ignored)
- serial_message[13] = (ignored)
- serial_message[14] = (ignored)
- serial_message[15] = (ignored)

The RS-232 server responds (response to remote request):

...usherbrooke.ca/.../index.php?title=...

Read all currents from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|---|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> bit 0-3 = 8 bit 4 = RTR = 0 bit 5 =1 (not used) bit 6 =1 (not used) |

Read all currents from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|--|---|--|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> CAN_ACTUATOR_CMD_SPEEDS=0x91 | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> CAN_MSG_DATA[0-1] <ul style="list-style-type: none"> LSB, MSB Speed motor 0 CAN_MSG_DATA[2-3] <ul style="list-style-type: none"> LSB, MSB Speed motor 1 CAN_MSG_DATA[4-5] <ul style="list-style-type: none"> LSB, MSB Speed motor 2 CAN_MSG_DATA[6-7] <ul style="list-style-type: none"> LSB, MSB Speed motor 3 |

Serial Message Received :

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x68
- serial_message[6] = 0x91
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = Motor 0 Speed LSB
- serial_message[9] = Motor 0 Speed MSB
- serial_message[10] = Motor 1 Speed LSB
- serial_message[11] = Motor 1 Speed MSB
- serial_message[12] = Motor 2 Speed LSB
- serial_message[13] = Motor 2 Speed MSB
- serial_message[14] = Motor 3 Speed LSB
- serial_message[15] = Motor 3 Speed MSB

Read all SetPoints from the motor controller

The computer sends this to the RS-232 server (Remote request) :

Read all setpoints from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|---|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> bit 0-3 = 8 bit 4 = RTR = 1 bit 5 =1 (not used) bit 6 =1 (not used) |

Read all setpoints from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|---|---|---|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> CAN_ACTUATOR_CMD_SETPOINTS=0x92 | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> CAN_MSG_DATA[0-7] (ignored) |

Serial Message Sent:

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x78
- serial_message[6] = 0x92
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = (ignored)
- serial_message[9] = (ignored)
- serial_message[10] = (ignored)
- serial_message[11] = (ignored)
- serial_message[12] = (ignored)
- serial_message[13] = (ignored)
- serial_message[14] = (ignored)
- serial_message[15] = (ignored)

The RS-232 server responds (response to remote request):

Read all setpoints from the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|---|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> header[0] = 'c' header[1] = 'a' header[2] = 'n' | <ul style="list-style-type: none"> CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> bit 0-3 = 8 bit 4 = RTR = 0 bit 5 =1 (not used) bit 6 =1 (not used) |

Read all setpoints from the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|---|---|--|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> CAN_ACTUATOR_CMD_SETPOINTS=0x92 | <ul style="list-style-type: none"> 0x01 = First controller 0x02 = Second controller | <ul style="list-style-type: none"> CAN_MSG_DATA[0-1] <ul style="list-style-type: none"> LSB, MSB SetPoint motor 0 CAN_MSG_DATA[2-3] <ul style="list-style-type: none"> LSB, MSB SetPoint motor 1 CAN_MSG_DATA[4-5] <ul style="list-style-type: none"> LSB, MSB SetPoint motor 2 CAN_MSG_DATA[6-7] <ul style="list-style-type: none"> LSB, MSB SetPoint motor 3 |

Serial Message Received :

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x68
- serial_message[6] = 0x92
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = Motor 0 SetPoint LSB
- serial_message[9] = Motor 0 SetPoint MSB
- serial_message[10] = Motor 1 SetPoint LSB
- serial_message[11] = Motor 1 SetPoint MSB
- serial_message[12] = Motor 2 SetPoint LSB
- serial_message[13] = Motor 2 SetPoint MSB
- serial_message[14] = Motor 3 SetPoint LSB
- serial_message[15] = Motor 3 SetPoint MSB

Write the SetPoint of motor [0-3]

The computer sends this to the RS-232 server (Remote request) :

Write a SetPoint to the motor controller RS-232 Data Format (16 bytes total) (part 1)

| HEADER | CAN_MSG_PRI_MSB_CMD | CAN_MSG_TYPE | CAN_MSG_BOOT_RTR_LENGTH |
|---|---|--|--|
| (3 bytes) | (1 byte) | (1 byte) | (1 byte) |
| <ul style="list-style-type: none"> ▪ header[0] = 'c' ▪ header[1] = 'a' ▪ header[2] = 'n' | <ul style="list-style-type: none"> ▪ CAN_MSG_PRI_MSB_CMD=0 | <ul style="list-style-type: none"> ▪ CAN_TYPE_ACTUATOR_HIGH_PRIORITY=0x02 | <ul style="list-style-type: none"> ▪ bit 0-3 = 3 ▪ bit 4 = RTR = 0 ▪ bit 5 = 1 (not used) ▪ bit 6 = 1 (not used) |

Write a SetPoint to the motor controller RS-232 Data Format (16 bytes total) (part 2)

| CAN_MSG_CMD | CAN_MSG_DEST | CAN_MSG_DATA |
|---|---|--|
| (1 byte) | (1 byte) | (8 bytes) |
| <ul style="list-style-type: none"> ▪ CAN_ACTUATOR_CMD_WRITE_SETPONT=0x93 | <ul style="list-style-type: none"> ▪ 0x01 = First controller ▪ 0x02 = Second controller | <ul style="list-style-type: none"> ▪ CAN_MSG_DATA[0] MOTOR_ID (0,1,2,3) ▪ CAN_MSG_DATA[1] SetPoint LSB ▪ CAN_MSG_DATA[2] SetPoint MSB |

Serial Message Sent:

- serial_message[0] = 'c'
- serial_message[1] = 'a'
- serial_message[2] = 'n'
- serial_message[3] = 0x00
- serial_message[4] = 0x02
- serial_message[5] = 0x63
- serial_message[6] = 0x93
- serial_message[7] = 0x01 or 0x02
- serial_message[8] = MOTOR_ID
- serial_message[9] = SetPoint LSB
- serial_message[10] = SetPoint MSB
- serial_message[11] = (ignored)
- serial_message[12] = (ignored)
- serial_message[13] = (ignored)
- serial_message[14] = (ignored)
- serial_message[15] = (ignored)

No serial message will be received

Additional Notes & Comments

- Single 12V power supply could be useful for direct connection to Pioneer robots. A 12V-->8.5V DC-DC (100W) converter is required for this operation. This item can be ordered at Vicor (<http://www.vicr.com/>) . 8.5V batteries would then be useless.
- Timing for all motors positions feedback can be obtained with the following calculation :
 - 2 serial messages (CMD_POSITIONS) * (1/57600 second/bits) * 16 bytes * 8 bits / byte) = 4.4 ms
 - Position refresh is then possible at maximum frequency : 225Hz.

Récupérée de « http://introlab.gel.usherbrooke.ca/mediawiki-introlab/index.php/Reddy_Update »